Fill-in reduction in sparse matrix factorizations using hypergraphs

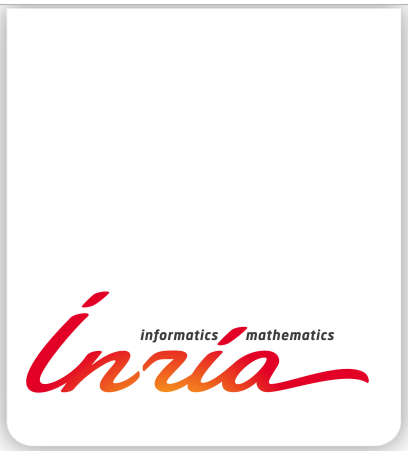
Oguz Kaya, Enver Kayaaslan, Bora Uçar, Iain S. Duff

Fill-in reduction in sparse matrix factorizations using hypergraphs

Oguz Kaya, Enver Kayaaslan, Bora Uçar, Iain S. Duff

RESEARCH REPORT
N° 8448
February 2013
Project-Team ROMA

ISSN 0249-6399 ISRN INRIA/RR--8448--FR+ENG



Fill-in reduction in sparse matrix factorizations using hypergraphs

Oguz Kaya*, Enver Kayaaslan†, Bora Uçar‡, Iain S. Duff§

Project-Team ROMA

Research Report n° 8448 — February 2013 — 24 pages

Abstract: We discuss the use of hypergraph partitioning-based methods for fill-reducing orderings of sparse matrices in Cholesky, LU and QR factorizations. For the Cholesky factorization, we investigate a recent result on pattern-wise decomposition of sparse matrices, generalize the result, and develop algorithmic tools to obtain more effective ordering methods. The generalized results help us formulate the fill-reducing ordering problem in LU factorization as we do in the Cholesky case, without ever symmetrizing the given matrix $\mathbf{A}$ as $|\mathbf{A}| + |\mathbf{A}^T|$ or $|\mathbf{A}^T||\mathbf{A}|$. For the QR factorization case, we adopt a recently proposed technique to use hypergraph models in a fairly standard manner. The method again does not form the possibly much denser matrix $|\mathbf{A}^T||\mathbf{A}|$. We also discuss alternatives in the LU and QR factorization cases where the symmetrized matrix can be used . We provide comparisons with the most common alternatives in all three cases.

Key-words: Fill-reducing orderings methods, Cholesky factorization, LU factorization, QR factorization, hypergraph partitioning, sparse matrices, direct methods.

* College of Computing (GATECH) Georgia Institute of Technology (Georgia Tech), Atlanta, GA

† INRIA and LIP, ENS Lyon, Frace

‡ CNRS and LIP, ENS Lyon, France

§ CERFACS, 42 Avenue Gaspard Coriolis, 31057, Toulouse, France and Building R 18, RAL, Oxon, OX11 0QX, England

RESEARCH CENTRE GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Réduction de remplissage dans les factorisation de matrices creuses avec des hypergraphes

Résumé : Nous discutons de l'utilisation de méthodes basées sur le partitionnement d'hypergraphes pour la réduction de remplissage des matrices creuses dans les factorisations Cholesky, LU et QR. Pour la factorisation de Cholesky, nous étudions un résultat récent sur la décomposition structurale des matrices creuses, généralisons le résultat, et développons des outils algorithmiques pour obtenir des méthodes plus efficaces. Les résultats génfalisés nous aident à formuler le problème de réduction de remplissage dans la factorisation LU comme nous le faisons dans le cas de Cholesky, sans jamais symétriser la matrice donnée $\mathbf{A}$ comme $|\mathbf{A}| + |\mathbf{A}^T|$ ou $|\mathbf{A}^T||\mathbf{A}|$. Pour le cas de la factorisation QR, nous adoptons une technique récemment proposé qui consiste à utiliser des modèles de hypergraphes d'une manière assez classique. La méthode ne construit pas l'éventuellement beaucoup plus dense matrice $|\mathbf{A}^T||\mathbf{A}|$. Nous discutons également des alternatives dans les cas de factorisation LU et QR où la matrice symétrisée peut être utilisé. Nous fournissons des comparaisons avec les méthodes les plus courantes dans les trois factorisations.

Mots-clés : méthodes de réduction de remplissage, factorisation de Cholesky, factorisation LU, factorisation QR, partitionnement des hypergraphes, matrices creuses, méthodes directes.

1 Introduction

We investigate the space requirement of Cholesky, LU and QR factorizations for sparse matrices. The Cholesky factorization of a positive definite matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ is given by $\mathbf{A} = \mathbf{L}\mathbf{L}^T$, where $\mathbf{L} \in \mathbb{R}^{n\times n}$ is a lower triangular matrix with positive diagonal entries. If $\mathbf{A}$ is square and admits an LU-factorization, then its LU-factorization is given by $\mathbf{A} = \mathbf{L}\mathbf{U}$, where $\mathbf{L}$ is lower triangular, and $\mathbf{U}$ is upper triangular. Let $\mathbf{A} \in \mathbb{R}^{m\times n}$ be a sparse rectangular matrix, $m \geq n$, with full column rank n. The QR factorization of $\mathbf{A}$ is given by $\mathbf{A} = \mathbf{Q}\mathbf{R}$, where $\mathbf{Q} \in \mathbb{R}^{m\times m}$ is an orthogonal matrix, and $\mathbf{R} \in \mathbb{R}^{m\times n}$ is an upper trapezoidal matrix. When $\mathbf{A}$ is sparse, these decompositions result in $\mathbf{L}, \mathbf{U}$ and $\mathbf{R}$ with nonzeros (called fill-ins) at positions that were originally zero in $\mathbf{A}$ or $\mathbf{A}^T\mathbf{A}$ (for QR). Our aim is to reduce the number of nonzeros in $\mathbf{L}, \mathbf{U}$ and $\mathbf{R}$ by permuting the nonzero structure of $\mathbf{A}$ into a special form and respecting this form when performing the reordering. In all three cases, we permute a matrix, say $\mathbf{M}$ which is not $\mathbf{A}$, into a form called singly bordered block diagonal form (SBBD) where the border consists of a set of columns. In this form, the removal of the border leaves a block diagonal matrix (the diagonal blocks are not necessarily square). Our contributions are on the definition of the matrix $\mathbf{M}$ for the three types of factorization and the algorithmic tools to construct $\mathbf{M}$. Once $\mathbf{M}$ is defined, we use a hypergraph partitioning algorithm to permute it into an SBBD form.

In Cholesky factorization, a symmetric permutation is applied to $\mathbf{A}$ to reduce the number of nonzeros in $\mathbf{L}$. There are many alternatives for finding such a permutation (see [16, Section 2.3.1] for a recent survey). To the best of our knowledge, all existing methods are based on the standard graph representation of a symmetric matrix, except the work by Çatalyürek et al. [7]. In this latter work, for a given $\mathbf{A}$, the authors find a sparse matrix $\mathbf{M}$ such that the nonzero pattern of $\mathbf{A}$ and $\mathbf{M}^T\mathbf{M}$ are identical (we use $\mathbf{A} \equiv \mathbf{M}^T\mathbf{M}$ to denote this equivalence). Then the matrix $\mathbf{M}$ is permuted into an SBBD form by a row and a column permutation. The column permutation of $\mathbf{M}$ is applied symmetrically to $\mathbf{A}$. Then, a variant of the well-known minimum degree algorithm [22] is used to finalize the ordering on $\mathbf{A}$. We first discuss an algorithm for finding such an $\mathbf{M}$ (by using the subroutine MC37 from the HSL Mathematical Software Library [26] that we describe in Section 3.1.1). This approach is a significant improvement over the existing scheme for obtaining $\mathbf{M}$, both in run time as well as the effectiveness of the resulting ordering for $\mathbf{A}$. Then we show that a similar approach can be followed if $\mathbf{A} \subseteq \mathbf{M}^T\mathbf{M}$ holds pattern-wise. In other words, the equality that was enforced in earlier work is a restriction. Without this restriction, we have more freedom to find a suitable $\mathbf{M}$. We exploit this freedom to devise another class of algorithms that are based on detecting clique-like structures in the graph of $\mathbf{A}$. This second class of algorithms run fast and, depending on a control parameter, improve the ordering quality or the run time of the tools that are used to find an SBBD form with respect to existing methods [7].

In LU factorization, the rows and the columns of $\mathbf{A}$ can be permuted nonsymmetrically to reduce the potential fill-in. Many ordering methods for LU factorization order the matrix $\mathbf{A}$ by using a column permutation, leaving the row permutation flexible for accommodating later numerical pivoting. In a sense, these methods minimize the fill in the Cholesky factorization of $\mathbf{A}^T\mathbf{A}$. Clearly, the fill in the Cholesky factor of $P_c\mathbf{A}^T\mathbf{A}P_c^T$ is independent of any

rowwise permutation P_r of $\mathbf{A}P_c$. Some LU factorization methods, such as SuperLU_DIST [32], use a static pivoting scheme during numerical factorization. In these methods, pivots are always chosen from the diagonal where very small ones are replaced by a larger value to avoid uncontrolled growth (or even break down) during factorization at the cost of numerical inaccuracy. When solving a linear system with such approximate factors, iterative refinement is used to recover the lost accuracy. In such a setting, ordering only the columns of $\mathbf{A}$ misses the opportunity to further reduce the fill-in, as no further pivoting will take place. We show that a structural decomposition of the form $\mathbf{A} \equiv \mathbf{C}^T\mathbf{B}$ can be used to permute $\mathbf{A}$ into a desirable form by permuting both $\mathbf{C}$ and $\mathbf{B}$ into an SBBD form. We relax this relation and show that as long as $\mathbf{A} \subseteq \mathbf{C}^T\mathbf{B}$, we can do the same, just as for the Cholesky factorization. We discuss the problem of finding the required structural decomposition and develop algorithms to tackle this problem.

In QR factorization, if the matrix $\mathbf{A}$ satisfies a condition called strong Hall (whose definition is given later in Section 2) then the fill in $\mathbf{R}$ can be reduced by ordering the columns of $\mathbf{A}$. This is because of the equivalence between the QR factorization of $\mathbf{A}$ and the Cholesky factorization of $\mathbf{A}^T\mathbf{A}$ [24, Theorem 5.2.2]. The general approach outlined for the Cholesky and LU factorizations can be followed to reduce the fill-in. $\mathbf{A}$ can be permuted to an SBBD form, thus defining a partial order of columns, and then the final ordering can be finalized on $\mathbf{A}$ using a variant of the minimum degree algorithm. We show that, within this framework, one needs to find a matrix $\mathbf{M}$ so that $\mathbf{A}^T\mathbf{A} \subseteq \mathbf{M}^T\mathbf{M}$, then find an SBBD form of $\mathbf{M}$ and finalize the ordering on $\mathbf{A}$. One way to proceed is to constrain $\mathbf{M}$ so that $\mathbf{M} \subseteq \mathbf{A}$, and $\mathbf{A}^T\mathbf{A} \equiv \mathbf{M}^T\mathbf{M}$. Here, we suggest the use of an algorithm from the literature [7, p. 2009]. Our contribution in ordering for QR factorization is therefore to show that certain tools from the literature can be used in a black box manner to effectively reduce the fill in the $\mathbf{R}$ factor. We note that our algorithm does not need the symmetrized matrix $|\mathbf{A}^T||\mathbf{A}|$. We share this property with Colamd [10], unlike graph partitioning-based approaches, such as using MeTiS [29] on $\mathbf{A}^T\mathbf{A}$.

The organization of the paper is as follows. We give background material on graphs, bipartite graphs, and hypergraphs in the following section. The ordering problem for each of the factorizations is formulated in separate subsections of Section 3. Then, in Section 4, we summarize some recent related work. The experimental investigations in which we compare the proposed new approaches with state-of-the-art standard methods are presented in Section 5. We conclude the paper by a summary in Section 6.

2 Background

2.1 Desirable matrix forms for factorization

We make use of two special forms of matrices described for an $m \times n$ sparse matrix $\mathbf{A}$ and an integer $K > 1$:

$$A_{DB} = \begin{pmatrix} A_{11} & & & & A_{1S} \\ & A_{22} & & & A_{2S} \\ & & \ddots & & \vdots \\ & & & A_{KK} & A_{KS} \\ A_{S_1} & A_{S2} & \cdots & A_{SK} & A_{SS} \end{pmatrix} \quad (1)$$

$$A_{SB} = \begin{pmatrix} A_{11} & & & A_{1S} \\ & A_{22} & & A_{2S} \\ & & \ddots & \vdots \\ & & A_{KK} & A_{KS} \end{pmatrix} \quad (2)$$

The first form A_{DB} (1) is called doubly bordered block-diagonal (DBBD) form. The second one A_{SB} (2) is called singly bordered block-diagonal (SBBD) form by columns.

An $m \times n$ matrix $\mathbf{A}$, with $m \geq n$, has strong Hall property by columns, if for every set C of columns with $|C| < n$, the number of rows which have at least one nonzero in the columns in C is at least $|C| + 1$. In particular, for a strong Hall matrix in SBBD form (2), each of the diagonal blocks (assuming they are non-empty) should have more rows than columns.

The relevance of these forms for Cholesky and LU factorization is that, if $\mathbf{A}$ is in a DBBD form, then the fill-in is confined to the nonzero blocks of $\mathbf{A}$, if the pivots are chosen first from the diagonal blocks, and then from the last block. Similarly in QR factorization, if $\mathbf{A}$ is in SBBD form, then $\mathbf{A}^T\mathbf{A}$ is in DBBD form; hence, the fill-in is confined to the nonzero diagonal and border blocks of $\mathbf{A}^T\mathbf{A}$. We note that using the structure of $\mathbf{A}^T\mathbf{A}$ to control the fill-in in $\mathbf{A}$ is a very well-known technique, see for example early work on sparse QR factorization algorithms [21, 23].

2.2 Graphs and bipartite graphs

The standard graph model $G = (V, E)$ corresponding to an $n \times n$ pattern symmetric matrix $\mathbf{A}$ has $|V| = n$ vertices, and an edge $(i, j) \in E$ for each off-diagonal nonzero pair $a_{ij} \neq 0$ (and $a_{ji} \neq 0$) in $\mathbf{A}$.

A bipartite graph $G = (U \cup V, E)$ has two sets of vertices U and V such that $E \subseteq U \times V$, i.e., all edges connect a vertex from U with a vertex from V. A bipartite graph G can be associated with a sparse matrix $\mathbf{B}$ so that $b_{ij} \neq 0$ if and only if $(i, j) \in E$ in the bipartite graph, for $u_i \in U$ and $v_j \in V$.

The edge-node incidence matrix $\mathbf{E}$ of a graph $G = (V, E)$, bipartite or undirected, has $|E|$ rows, each corresponding to a unique edge and $|V|$ columns, each corresponding to a unique vertex. A row r of $\mathbf{E}$ corresponding to the edge (u, v) has two nonzeros, one in the column corresponding to the vertex u and another in the column corresponding to the vertex v.

2.3 Hypergraphs and hypergraph partitioning

A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ is defined as a set of vertices $\mathcal{V}$ and a set of nets (or hyperedges) $\mathcal{E}$. Every net $n_j \in \mathcal{E}$ is a subset of vertices, i.e., $n_j \subseteq \mathcal{V}$. Weights can be associated with the vertices. We use $w(v)$ to denote the weight of the vertex v and extend this notation to a set of vertices $\mathcal{S}$ as $W(\mathcal{S}) = \sum_{v \in \mathcal{S}} w(v)$. In all hypergraph models in this paper, we use unit vertex weights.

Given a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, $\Pi = \{\mathcal{V}_1, \ldots, \mathcal{V}_K\}$ is called a K-way partition of the vertex set $\mathcal{V}$ if each part is nonempty, i.e., $\mathcal{V}_k \neq \emptyset$ for $1 \leq k \leq K$; parts are pairwise disjoint, i.e., $\mathcal{V}_k \cap \mathcal{V}_\ell = \emptyset$ for $1 \leq k < \ell \leq K$; and the union

of parts gives $\mathcal{V}$, i.e., $\bigcup_k \mathcal{V}_k = \mathcal{V}$. For a given K-way vertex partition Π, let $W_{avg} = W(V)/K$ and $W_{max} = \max_k\{W(\mathcal{V}_k)\}$ denote the average and the maximum weight of a part, respectively. Π is then said to be balanced for a given $\varepsilon \geq 0$, if

$$\frac{W_{max}}{W_{avg}} \leq (1+\varepsilon)\ . \tag{3}$$

In a partition Π of $\mathcal{H}$, a net that has at least one vertex in a part is said to *connect* that part. The *connectivity set* Λ_j of a net n_j is defined as the set of parts connected by n_j. The *connectivity* $\lambda_j = |\Lambda_j|$ of a net n_j denotes the number of parts connected by n_j. A net n_j is said to be *cut* (*external*) if it connects more than one part (i.e., $\lambda_j > 1$) and *uncut* (*internal*) otherwise (i.e., $\lambda_j = 1$). The set of external nets of a partition Π is denoted $\mathcal{E}_E$. The partitioning objective is to minimize a function called *cutsize* defined over the cut nets. The relevant definition of the cutsize function for our purposes in this paper is called the *cut-net* metric:

$$cutsize(\Pi) \quad = \quad \sum_{n_j \in \mathcal{E}_E} 1\ , \tag{4}$$

In the cut-net metric (4), each cut net contributes one to the cutsize. Sometimes costs are associated with the nets, in which case those costs enter as a factor into equation (4). For our purposes in this paper, we do not associate costs with nets and just use the above cutsize definition. The hypergraph partitioning problem can be defined as the task of dividing the vertices of a hypergraph into K parts so that the cutsize is minimized, while a balance criterion (3) is met for a given ε. The hypergraph partitioning problem is known to be NP-hard [31].

2.4 The column-net hypergraph model

We use the column-net hypergraph model [5] of sparse matrices. The column-net hypergraph model $\mathcal{H}_\mathcal{C} = (\mathcal{R}, \mathcal{C})$ of an $m \times n$ sparse matrix $\mathbf{A}$ has m vertices and n nets. Each vertex in $\mathcal{R}$ corresponds to a row of $\mathbf{A}$. Similarly, each net in $\mathcal{C}$ corresponds to a column of $\mathbf{A}$. Furthermore, for a vertex r_i and net c_j, $r_i \in c_j$ if and only if $a_{ij} \neq 0$. Each vertex has unit weight. A K-way partition $\Pi = \{\mathcal{V}_1, \ldots, \mathcal{V}_K\}$ of the column-net model of a sparse matrix $\mathbf{A}$ can be used to permute $\mathbf{A}$ into a singly-bordered form

$$P_r \mathbf{A} P_c^T = \begin{pmatrix} \mathbf{A}_{11} & & & \mathbf{A}_{1C} \\ & \mathbf{A}_{22} & & \mathbf{A}_{2C} \\ & & \ddots & \vdots \\ & & \mathbf{A}_{KK} & \mathbf{A}_{KC} \end{pmatrix}, \tag{5}$$

where P_r and P_c are permutation matrices [3]. P_r permutes the rows of $\mathbf{A}$ so that the rows corresponding to the vertices in part $\mathcal{V}_i$ come before those in part $\mathcal{V}_j$ for $1 \leq i < j \leq K$. P_c permutes the columns corresponding to the nets internal to part $\mathcal{V}_i$ before the columns corresponding to the nets internal to part $\mathcal{V}_j$ for $1 \leq i < j \leq K$, and permutes the columns corresponding to the cut nets (external nets) to the end. Clearly, the border size is equal to the number of cut nets as measured by the cutsize function (4).

3 Problems and algorithms

For the three standard factorizations, we will define a suitable matrix $\mathbf{M}$. The matrix $\mathbf{M}$ will then be permuted to an SBBD form, using the column-net hypergraph model which will result in a DBBD form for the matrix $\mathbf{M}^T\mathbf{M}$ or any subset of it, in particular for $\mathbf{A} \subseteq \mathbf{M}^T\mathbf{M}$ or $\mathbf{A}^T\mathbf{A} \subseteq \mathbf{M}^T\mathbf{M}$. Then, the final ordering on the given matrix $\mathbf{A}$ will be obtained by using a suitable variant of the minimum degree algorithm.

3.1 Cholesky factorization

Let $\mathbf{A}$ be a pattern symmetric matrix with a zero-free diagonal, and let $\mathbf{M}$ be a matrix such that $\mathbf{A} \equiv \mathbf{M}^T\mathbf{M}$ holds pattern-wise. Suppose that the column-net hypergraph model of $\mathbf{M}$ is partitioned to obtain $\mathbf{M}_{SB} = P_r\mathbf{M}P_c$ as in (5). Çatalyürek et al. [7] show that P_c can be used effectively to permute $\mathbf{A}$ into a DBBD form. We restate this as a theorem.

Theorem 1 ([7])**.** *Let $\boldsymbol{M}$ be a matrix in a singly bordered block diagonal form by columns. Then $\boldsymbol{A} \equiv \boldsymbol{M}^T\boldsymbol{M}$ is in doubly bordered block diagonal form.*

This structural decomposition-based formulation has some advantages [7, p. 2000] over the graph partitioning-based algorithms used in current state-of-the-art partitioning methods. In this work, we generalize the structural decomposition formulation restated in Theorem 1 with the following theorem.

Theorem 2. *Let$\boldsymbol{A}_{n\times n}$ and $\boldsymbol{M}_{n\times n}$ be two matrices where $\boldsymbol{A}$ is pattern symmetric and $\boldsymbol{A} \subseteq \boldsymbol{M}^T\boldsymbol{M}$. Let P_r and P_c be two permutation matrices such that $\boldsymbol{M}_{SB} = P_r\boldsymbol{M}P_c^T$ is in singly bordered block diagonal form by columns. Then $P_c\boldsymbol{A}P_c^T$ is in doubly bordered block diagonal form.*

Proof. Since the nonzero pattern of $\mathbf{A}$ is a subset of $\mathbf{M}^T\mathbf{M}$, for permutation matrices P_c and P_r it holds that $P_c\mathbf{A}P_c^T \subseteq P_c\mathbf{M}^T\mathbf{M}P_c^T = P_c\mathbf{M}^TP_r^TP_r\mathbf{M}P_c^T = \mathbf{M}_{SB}^T\mathbf{M}_{SB}$. Theorem 1 implies that $\mathbf{M}_{SB}^T\mathbf{M}_{SB}$ is in DBBD form. Hence, $P_c\mathbf{A}P_c^T$ is also in DBBD form since $P_c\mathbf{A}P_c^T \subseteq \mathbf{M}_{SB}^T\mathbf{M}_{SB}$. □

The theorem essentially says that for a given $\mathbf{A}$, we can find an $\mathbf{M}$ that is different from a matrix resulting from an exact structural decomposition of $\mathbf{A}$. Our aim is to exploit this freedom to reduce the cost of the hypergraph partitioning algorithm. This can be achieved by having a small number of rows and a small number of nonzeros in $\mathbf{M}$. However, we need to ensure that $\mathbf{M}^T\mathbf{M}$ is not very far from $\mathbf{A}$; for example, one condition may be that $\mathbf{M}^T\mathbf{M}\setminus\mathbf{A}$ should not contain many entries. We now discuss some methods for constructing $\mathbf{M}$.

3.1.1 Existing solutions.

First, we could choose $\mathbf{M}$ to be the edge-node incidence matrix of the graph of $\mathbf{A}$. This implies that $\mathbf{M}^T\mathbf{M} \setminus \mathbf{A} = \emptyset$. However, $\mathbf{M}$ will have $nnz(tril(\mathbf{A})) - n$ rows and twice as many nonzeros as $\mathbf{A}$. Thus, even if the algorithm to create the edge-node incidence matrix $\mathbf{M}$ from $\mathbf{A}$ can be done in $\mathcal{O}(n + \tau)$ time, one cannot expect much from this formulation.

Çatalyürek et al. [7] present algorithms to construct an $\mathbf{M}$ for a given (pattern symmetric) $\mathbf{A}$. Their objective is to minimize the number of rows in $\mathbf{M}$

while having $\mathbf{M}^T\mathbf{M} \setminus \mathbf{A} = \emptyset$. They show that this is equivalent to finding a minimum edge clique cover of a graph, which is an NP-complete problem. In order to develop efficient algorithms, they restrict the cliques to be of maximum size ℓ for some $\ell \geq 3$, where $\ell = 2$ corresponds to using the edge-node incidence matrix. The algorithm requires $\mathcal{O}(n\Delta^\ell)$ time and $\mathcal{O}(n\Delta^{\ell-1})$ space, and creates cliques of size 2-to-ℓ, where Δ is the maximum degree of a vertex. Çatalyürek et al. [7] recommend $\ell = 3$ or $\ell = 4$ for practical problems.

We propose the use of MC37 from HSL [26] as an alternative method. In MC37, a rowwise representation of the lower triangular part of the matrix is first formed, with column indices in order within the rows. MC37 then proceeds greedily. It visits the rows of this sorted matrix in reverse order. At a row, the nonzero entries that are not already covered by the existing cliques are traversed. Any nonzero entry either adds the corresponding column to the current clique, or starts a new clique (with the current row). This way MC37 builds as large cliques as it can. When the cliques covering the nonzeros in a row are constructed, the nonzero entries (in other rows) contained in those cliques are marked as covered. At the end, the rows of $\mathbf{M}$ correspond to the cliques that are identified, where the nonzeros in a row of $\mathbf{M}$ correspond to the vertices in the associated clique. MC37 guarantees that $\mathbf{M}^T\mathbf{M} \setminus \mathbf{A} = \emptyset$. The algorithm runs in $\mathcal{O}(\sum r_i^2)$ where r_i is the number of nonzeros in the ith row of the lower triangular part of $\mathbf{A}$. MC37 can find larger cliques than the algorithm of Çatalyürek et al. [7].

3.1.2 Proposed method: Covering with quasi-cliques.

The algorithm we propose is a generalization of the algorithm implemented in MC37. Instead of finding cliques, we find sets of vertices that are close to being a clique to cover the edges of the standard undirected graph model of $\mathbf{A}$. More formally, for a given $\beta > 0$, a set of vertices $S \subseteq V$ is called a β-quasi-clique (or β-clique for short) of a graph $G = (V, E)$, if $\frac{|S \times S \cap E|}{|S|(|S|-1)/2} \geq \beta$. Once the quasi-cliques are found, the matrix $\mathbf{M}$ can be constructed as before (its rows correspond to quasi-cliques and its columns correspond to vertices in those cliques). Our aim is to cover the nonzeros with the minimum number of β-cliques. The optimization problem that we pose is the following, where a nomenclature common in the computer science literature is used, see for example [2].

Minimum Quasi-Clique Cover (MQCC)
Instance: An undirected graph $G = (V, E)$ and a fixed real number β such that $0 < \beta < 1$.
Solution: A β-quasi-clique cover $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\}$, where each $\mathcal{C}_i$ is a β-quasi-clique.
Measure: Cardinality k of the cover, i.e., the number of β-quasi-cliques $\mathcal{C}_i$.

Kaya et al. [30] show that the Minimum Quasi-Clique Cover problem is NP-complete. We therefore propose a heuristic algorithm, shown in Algorithm 1.

The proposed heuristic QCC performs a number of iterations (the **while** loop). At each iteration the algorithm grows a quasi-clique (the **repeat-until** loop) using uncovered edges in the graph as long as the edge density of the current quasi-clique is above the given number β. After a quasi-clique is formed, covered edges are removed from the graph before starting the next iteration.

Input: An undirected graph $G = (V, E)$
Output: $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\}$: a β-quasi clique cover $\mathcal{C}$ with k elements
  $\mathcal{C} \leftarrow \emptyset$; $i \leftarrow 1$; score$(v) \leftarrow 0$ for all $v \in V$ ▶ initialization
  while $E \neq \emptyset$ **do**
    $v \leftarrow$ a node with maximum degree ▶ v is the seed of a clique
    $\mathcal{C}_i \leftarrow \emptyset$
    $B \leftarrow \emptyset$ ▶ vertices having a neighbour in quasi-clique $\mathcal{C}_i$
    repeat
      $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \{v\}$
      for each $u \in adj(v), u \notin \mathcal{C}_i$ **do**
        score$(u) \leftarrow$ score$(u) + 1$
        $B \leftarrow B \cup u$ ▶ u is not added twice to B
      end for
      $v \leftarrow$ a vertex from B with maximum score
    until $2\frac{|E(\mathcal{C}_i)| + \text{score}(v)}{|\mathcal{C}_i|(|\mathcal{C}_i|+1)} < \beta$
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathcal{C}_i\}$
    $E \leftarrow E \setminus \mathcal{C}_i \times \mathcal{C}_i$ ▶ purge also the adjacency list of vertices in $\mathcal{C}_i$
    score$(v) \leftarrow 0$ for all $v \in B$
    $i \leftarrow i + 1$
  end while

Algorithm 1: QCC(G, β)

At each step of greedily growing the current quasi-clique, a vertex with the maximum score (which is the number of neighbours in the current quasi-clique) is added to the quasi-clique. We use the following two tie-breaking strategies if more than one vertex has the maximum score. The first strategy picks the vertex with the smallest degree (denoted by SF). The motivation behind this strategy is that it is relatively harder for a small degree vertex to have a high connectivity to a quasi-clique. Therefore, whenever a tie occurs, to avoid having to cover the edges incident on such a vertex with small quasi-cliques, it might be preferable to cover these edges by adding the small degree vertex to the quasi-clique. The second strategy breaks the ties by picking the vertex with the largest degree (denoted by LF) in order to maximize the connectivity of the potential vertex candidates for the current quasi-clique. Thereby, it aims to form larger quasi-cliques.

The run time of the proposed QCC heuristic is $\mathcal{O}(\sum d_i^2)$ where d_i is the degree of the vertex v_i. This is a pessimistic estimate and is based on the following observation. A vertex v_i can be added to d_i quasi-cliques and updating the score of its neighbours with respect to the most recent quasi-clique requires $\mathcal{O}(d_i)$ time. Summing over all vertices yields the desired result. This will rarely be attained in practice as once a quasi-clique is formed, many of the neighbours of a vertex will appear in that clique and the cardinality of the edge set will reduce significantly. On the other hand, the formula suggests that we should be careful when there are some high degree vertices. In that case, many of the cliques containing those vertices will be small in size and the repeated score updates will increase the run time. We therefore handle high degree vertices separately in our practical algorithm for matrix ordering (by removing those vertices from the graph of $\mathbf{A}$ so that the quasi-clique cover algorithms do not

need to cover the incident edges). Depending on β, the number of quasi-cliques (i.e., the number of rows of $\mathbf{M}$) found by the proposed heuristic is more likely to be less than the number of cliques found by MC37. Similarly, the total size of the quasi-cliques (i.e., the number of nonzeros in $\mathbf{M}$) is more likely to be less than the total size of the cliques found by MC37.

3.2 LU factorization

Consider an LU factorization of a (pattern) unsymmetric matrix $\mathbf{A}$ with a static pivoting strategy. In this case again, it is desirable to put $\mathbf{A}$ into doubly bordered block diagonal form to confine the fill-in to the nonzero blocks. Since $\mathbf{A}$ is unsymmetric, the structural decomposition schemes described in Theorems 1 and 2 are not relevant. The required decomposition is described by the following theorem.

Theorem 3. *Let $\boldsymbol{A}_{n\times n}$, $\boldsymbol{B}_{m\times n}$ and $\boldsymbol{C}_{m\times n}$ be three matrices so that $\boldsymbol{A} \equiv \boldsymbol{C}^T\boldsymbol{B}$ holds. Let $\boldsymbol{M} \equiv \boldsymbol{B} + \boldsymbol{C}$ be the union of nonzero patterns of $\boldsymbol{B}$ and $\boldsymbol{C}$. Also, let P_r and P_c be two permutation matrices such that $P_r\boldsymbol{M}P_c^T$ is in singly bordered block diagonal form. Then, $P_c\boldsymbol{A}P_c^T$ is in doubly bordered block diagonal form.*

Proof. Clearly, $\mathbf{C}^T\mathbf{B} \subseteq \mathbf{M}^T\mathbf{M}$, since the nonzero structure of $\mathbf{M}$ is the union of that of $\mathbf{B}$ and $\mathbf{C}$. Theorem 1 implies that if $P_r\mathbf{M}P_c^T$ is in SBBD form, then $P_c\mathbf{M}^T\mathbf{M}P_c^T$ is in DBBD form. Since $\mathbf{A} \equiv \mathbf{B}^T\mathbf{C} \subseteq \mathbf{M}^T\mathbf{M}$, Theorem 2 implies that $P_c\mathbf{A}P_c^T$ is in DBBD form. □

Notice that Theorem 3 is a generalization of Theorem 1. In particular, for a symmetric $\mathbf{A}$, one can take $\mathbf{B} \equiv \mathbf{C}$ and recover Theorem 1. However, by only requiring that $\mathbf{C}^T\mathbf{B} = \mathbf{B}^T\mathbf{C}$, other kinds of structural decompositions for a symmetric matrix can be devised — we have not investigated this possibility yet.

As we did before, we can relax the equivalence constraint in the above theorem. We state this as a theorem.

Theorem 4. *Let $\boldsymbol{A}_{n\times n}$, $\boldsymbol{B}_{m\times n}$ and $\boldsymbol{C}_{m\times n}$ be three matrices such that $\boldsymbol{A} \subseteq \boldsymbol{C}^T\boldsymbol{B}$. Let P_r and P_c be two permutation matrices such that $P_r\boldsymbol{M}P_c^T$ where $\boldsymbol{M} \equiv \boldsymbol{B} + \boldsymbol{C}$ is in singly bordered block diagonal form by columns. Then $P_c\boldsymbol{A}P_c^T$ is in doubly bordered block diagonal form.*

Proof. Since $\mathbf{A} \subseteq \mathbf{C}^T\mathbf{B} \subseteq \mathbf{M}^T\mathbf{M}$, Theorem 2 implies that any permutation matrix P_c that puts $P_r\mathbf{M}P_c^T$ in SBBD form also puts $P_c\mathbf{A}P_c^T$ in DBBD form. □

Theorem 4 again highlights the freedom available for finding the matrices $\mathbf{B}$ and $\mathbf{C}$. After discussing some existing solutions for the decomposition in Theorem 3, we will propose two methods which take advantage of the inequality in Theorem 4.

3.2.1 Existing solutions.

First, consider the $\mathbf{A} \equiv \mathbf{C}^T\mathbf{B}$ case. Here we want to construct such a $\mathbf{C}$ and a $\mathbf{B}$ that the ith column of $\mathbf{C}$ has nonempty inner products with those columns of $\mathbf{B}$ which are indexed by the nonzero columns in the ith row of $\mathbf{A}$. This can be best understood with the help of bipartite graphs, which we discuss below.

A biclique (R, C) in a bipartite graph $G_B = (U \cup V, E)$ contains two sets of vertices $R \subseteq U$ and $C \subseteq V$ such that for all $r \in R$ and $c \in C$, we have the edge (r, c) in E. If we have a set of bicliques $\mathcal{B}$ covering all edges of the bipartite graph of a square matrix $\mathbf{A}$, then we can construct the matrices $\mathbf{C}$ and $\mathbf{B}$ both with $|\mathcal{B}|$ rows and n columns as follows. For each biclique (R, C), we have a row in $\mathbf{C}$ containing nonzeros in the columns corresponding to those rows of $\mathbf{A}$ in R, and we have a corresponding row in $\mathbf{B}$ containing nonzeros in the columns corresponding to those columns of $\mathbf{A}$ in C.

Clearly, such a structural decomposition exists, as we can take $\mathbf{C} \equiv \mathbf{I}$ and set $\mathbf{B} \equiv \mathbf{A}$. In order to see the relationship with the bipartite graph $G_B = (U \cup V, E)$ and its bicliques, we note that the decomposition $\mathbf{C} \equiv \mathbf{I}$ and $\mathbf{B} \equiv \mathbf{A}$ corresponds to using a biclique containing a single column vertex with all of its neighbouring row vertices. In general, however, one should search for a smaller number of bicliques. The underlying problem is known as the minimum biclique cover (MBC) problem which is NP-hard [34] and can be stated as follows.

Minimum Biclique Cover (MBC)
Instance: A bipartite graph $G = (U \cup V, E)$.
Solution: A biclique cover $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\}$, where each $\mathcal{C}_i$ induces a biclique.
Measure: Cardinality k of the cover, i.e., the number of bicliques $\mathcal{C}_i$.

Ene et al. [18] propose an exponential time exact algorithm for MBC problem which turns out to be practical for their problems. They also propose a polynomial time greedy algorithm. The greedy algorithm constructs one biclique at a time by choosing a row vertex r and all of its neighbours $adj(r)$, and then adds all other row vertices that are adjacent to all column vertices in $adj(r)$ to the biclique. A number of criteria are used to select the first vertex r: fewest uncovered incident edges, most uncovered incident edges, and a random available vertex (that has some uncovered incident edges). Ene et al. found that the first criterion is better than the others.

3.2.2 Proposed methods I: Covering with quasi-bicliques.

The above solutions are applicable for Theorem 3. We formalize the underlying problem for Theorem 4, based on quasi-bicliques that are analogous to quasi-cliques in undirected graphs.

Minimum Quasi-Biclique Cover (MQBC)
Instance: A bipartite graph $G = (U \cup V, E)$ and a fixed real number β such that $0 < \beta < 1$.
Solution: A β-quasi-biclique cover $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\}$, where each $\mathcal{C}_i$ is a β-quasi-biclique.
Measure: Cardinality k of the cover, i.e., the number of β-quasi-bicliques $\mathcal{C}_i$.

Once the β-cliques are found, the matrices $\mathbf{B}$ and $\mathbf{C}$ can be constructed as before. The MQBC problem seems to be as hard as the MQCC problem. We therefore propose a heuristic algorithm for the MQBC problem. The algorithm is similar in structure to the proposed QCC algorithm and grows one β-quasi-biclique at a time. Since the underlying graph is bipartite we need to adapt

Algorithm 1 in the following ways: (i) the boundary vertex set of a biclique is also bipartite; (ii) the score is updated for one set (row or column vertices) of boundary vertices; (ii) the clique density formula in the **repeat-until** loop depends on the type of the vertex with the maximum score (a column vertex c which has a smaller score than a row vertex r can result in a denser biclique than r does). These differences necessitate keeping the row and column vertices separately. As in the QCC algorithm, each vertex v_i in the bipartite graph with a degree d_i can appear at most d_i times in a quasi-biclique, and each time $O(d_i)$ time can be spent in updating the scores of its neighbours, yielding a worst-case run time complexity of $O(\sum d_i^2)$.

3.2.3 Proposed methods II: Exploiting symmetrization.

Our second proposal is to make use of the methods discussed in Section 3.1.1. Consider the structural decomposition of the symmetrized matrix $\mathbf{A} + \mathbf{A}^T \equiv \mathbf{M}^T\mathbf{M}$ using any of the methods discussed in Section 3.1.1. For such an $\mathbf{M}$, we can find $\mathbf{C}$ and $\mathbf{B}$ such that $\mathbf{M} \equiv \mathbf{C} + \mathbf{B}$, and $\mathbf{A} \subseteq \mathbf{C}^T\mathbf{B}$ (in particular we can take $\mathbf{C} \equiv \mathbf{B} \equiv \mathbf{M}$). Since we do not need the individual matrices $\mathbf{B}$ and $\mathbf{C}$, but their sum, having $\mathbf{M}$ is enough for fill-reducing purposes. That is, the structural decomposition methods used in the Cholesky case when applied to $\mathbf{A} + \mathbf{A}^T$ compute a matrix $\mathbf{M} \equiv \mathbf{C} + \mathbf{B}$ such that $\mathbf{A} \subseteq \mathbf{M}^T\mathbf{M}$, without finding the individual matrices $\mathbf{B}$ and $\mathbf{C}$ described in Theorem 4.

3.3 QR factorization

Consider the QR factorization of an $m \times n$ matrix $\mathbf{A}$ with $m \geq n$ having the strong Hall property by columns. The R-factor $\mathbf{R}$ is equal to the Cholesky factor of $\mathbf{A}^T\mathbf{A}$. One can permute $\mathbf{A}$ into an SBBD form by columns (so that $\mathbf{A}^T\mathbf{A}$ is in doubly bordered form) to restrict the fill-in to certain regions and finalize the ordering within blocks using a variant of the minimum degree heuristic. As discussed before, $\mathbf{A}$ can be permuted into SBBD form using hypergraph partitioning methods [3]. In this case also we can find a better matrix $\mathbf{M}$ to permute $\mathbf{A}$ into SBBD form. We first start with a corollary describing one such $\mathbf{M}$.

Corollary 1 (to Theorem 1)**.** *Let $\boldsymbol{A}_{m\times n}$ and $\boldsymbol{M}_{p\times n}$ be two matrices such that $\boldsymbol{A}^T\boldsymbol{A} \equiv \boldsymbol{M}^T\boldsymbol{M}$. Let P_r and P_c be two permutation matrices such that $P_r\boldsymbol{M}P_c^T$ is in singly bordered block diagonal form by columns. Then $P_c\boldsymbol{A}^T\boldsymbol{A}P_c^T$ is in doubly bordered block diagonal form.*

The corollary is easy to establish using Theorem 1 by considering the matrix $\mathbf{B} \equiv \mathbf{A}^T\mathbf{A}$ and its structural decomposition $\mathbf{B} \equiv \mathbf{M}^T\mathbf{M}$. As we have done before, the equivalence constraint can be relaxed. We state this as a corollary to Theorem 2.

Corollary 2 (to Theorem 2)**.** *Let $\boldsymbol{A}_{m\times n}$ and $\boldsymbol{M}_{p\times n}$ be two matrices such that $\boldsymbol{A}^T\boldsymbol{A} \subseteq \boldsymbol{M}^T\boldsymbol{M}$. Let P_r and P_c be two permutation matrices such that $P_r\boldsymbol{M}P_c^T$ is in singly bordered block diagonal form by columns. Then $P_c\boldsymbol{A}^T\boldsymbol{A}P_c^T$ is in doubly bordered block diagonal form.*

The proof of the corollary can again be obtained by considering the matrix $\mathbf{B} \equiv \mathbf{A}^T\mathbf{A}$ and its super set $\mathbf{B} \subseteq \mathbf{M}^T\mathbf{M}$ as in Theorem 2.

It can be seen that $\mathbf{A}$ can be permuted into singly bordered block diagonal form using the column permutation P_c and a row permutation. In the singly bordered form $\mathbf{A}_{SB}$, each diagonal block should have at least as many rows as columns for the QR factorization to respect the predicted fill-in correctly. Since we assumed the strong Hall property by columns, this condition is satisfied a priori.

As before, a good $\mathbf{M}$ should have a small number of rows and a small number of nonzeros, and $\mathbf{M}^T\mathbf{M}$ should not be far from $\mathbf{A}^T\mathbf{A}$. One way to guarantee this is to consider the set of matrices whose sparsity pattern is a subset of that of $\mathbf{A}$. Çatalyürek et al. [7] propose a method called sparsification to find a matrix $\mathbf{M}$ by deleting nonzeros from $\mathbf{A}$ (they use sparsification for Cholesky factorization). The essential idea is to check nonzeros in each column j one by one to see if they are necessary to have $\mathbf{M}^T\mathbf{M} \equiv \mathbf{A}^T\mathbf{A}$, and if so to copy those entries to $\mathbf{M}$. This is done by considering the vertices in all cliques containing the given column j. If the vertices in a clique, say i corresponding to row i, appear in other cliques containing j, the membership of j in the first one can be discarded by not copying the nonzero entry a_{ij} to $\mathbf{M}$. At a column, the discarded entries should be taken into account while processing nonzeros. The overall complexity of the algorithm is $\mathcal{O}(\sum_i |r_i|^2)$, where $|r_i|$ denotes the number of nonzeros in row i of $\mathbf{A}$. A similar algorithm is discussed elsewhere [19, Section 4].

4 Some related work on the nested dissection ordering

As we said earlier, our theoretical findings generalize those of Aykanat et al. [7] for the Cholesky factorization. We now summarize some other recent related work based on the nested dissection ordering and highlight our contributions with respect to them.

Brainman and Toledo [4] discuss a nested dissection based method to minimize fill-in in the LU factorization with partial pivoting. They find a column ordering Q for $\mathbf{A}$ which reduces the fill-in in the Cholesky factor of $Q^T\mathbf{A}^T\mathbf{A}Q$. This way they exploit the fact that the effect of any row permutation on the fill-in is accounted for. The proposed method does not form $\mathbf{A}^T\mathbf{A}$. First a separator is found for the graph of $\mathbf{A}+\mathbf{A}^T$ using standard tools (such as MeTiS). The separator is then modified to be a separator for $\mathbf{A}^T\mathbf{A}$. Our approach in finding a singly bordered form for $\mathbf{A}$ is similar in the sense that the size of the separator in $\mathbf{A}^T\mathbf{A}$ is reduced without ever forming the product $\mathbf{A}^T\mathbf{A}$. Our approach is more direct in the sense that the objective of the partitioning is to reduce the size of the border (in other words, the size of the separator in $\mathbf{A}^T\mathbf{A}$).

Aykanat et al. [3] discuss methods to obtain singly bordered block diagonal forms for arbitrary matrices. Their motivation is for load balancing in parallel factorization, where the size of the border corresponds to the size of the serial subproblem. They use hypergraphs where the hypergraph partitioning function corresponds to the size of the border. We show that this approach is also useful in reducing fill-in.

Hu and Scott [27] obtain a singly bordered block diagonal form for square matrices by finding vertex separators in the graph of $\mathbf{A}+\mathbf{A}^T$ by combining earlier ideas [4, 28]. Duff and Scott [14] exploit such forms to develop an efficient

parallel unsymmetric LU factorization based solver.

Grigori et al. [25] discuss hypergraph partitioning models in the spirit of Brainman and Toledo's approach. That is, they obtain a singly bordered block diagonal form for $\mathbf{A}$ which corresponds to a doubly bordered block diagonal form for $\mathbf{A}^T\mathbf{A}$. We show that, in this case, sparsification helps to reduce the run time and also reduces the cutsize.

Fagginger Auer and Bisseling [20] use geometric information associated with a matrix (or create that information automatically) to permute a given square matrix into doubly bordered block diagonal form with two diagonal blocks (and a border). Then each diagonal block is recursively partitioned into two. The overall approach is geared towards GPU-like systems having many-cores with shared memory.

The proposed ordering methods for LU or indeed any ordering methods based solely on matrix structure are particularly suitable when performing LU factorization with static pivoting. This is the numerical factorization scheme used for example by SuperLU_DIST [32]. In such LU factorization methods, a column permutation is first found (e.g., MC64's maximum product algorithm) so that the resulting matrix has large entries on its diagonal; then the diagonal entries are used as pivots during factorization (no further pivoting will take place). In this case, permuting $\mathbf{A}$ to doubly bordered block diagonal form so as to confine the fill-in to the nonzero blocks is a good way to control it. Methods based on singly bordered form, or in general those that are based on the pattern of $\mathbf{A}^T\mathbf{A}$, would in general result in much more fill-in when performing LU factorization with static pivoting. Of course, the proposed ordering methods can be used with LU factorization methods that perform pivoting. However, in this case the effectiveness of the proposed methods cannot be easily evaluated.

5 Experiments

We present the experimental results in three subsections, each concerned with one of the factorizations. We summarize the findings in a final subsection. The experimental set up and data sets are different for the three factorizations and therefore are described in the corresponding subsection. Some parts of the set up are common. We use PaToH [6] through its Matlab interface [35, 36] for hypergraph partitioning. All matrices are from the UFL collection [9]. Since PaToH uses randomized algorithms, we run each ordering algorithm (that uses PaToH as a partitioner) five times and report the average values in the following. In all our experiments, we use PaToH with the setting "quality". This setting improves the quality of PaToH's results although with increased run time. This way we demonstrate better the effectiveness of the structural decomposition methods and the importance of partitioning. In preliminary experiments [17], we used PaToH with the setting "default". The "quality" setting improved the resulting fill-in uniformly in all experiments by around 3% for all our structural decomposition methods.

5.1 Cholesky factorization

The matrices used in our experiments are chosen to satisfy the following properties. They are square, of order greater than 70000, have at least 2.5 nonzeros

per row on average, and have at most 20 million nonzeros. These properties ensure that the matrices are not too small, and are not close to being diagonal, but are small enough to be run using Matlab. The properties enable an automatic selection of a set of matrices from diverse application domains without having to specify each individually. Because the current UFL index does not contain much fill-in information for matrices, we used an older index of the collection which had 2328 matrices. 222 of these matrices have the properties that we just described. For matrices with unsymmetric patterns, we used the symmetrized matrix $\mathbf{A} + \mathbf{A}^T$. We made the diagonal of $\mathbf{A}$ zero-free. After this preprocessing, some of the matrices were reducible, and there were some with the same pattern. We discarded the reducible ones and kept only one matrix from a set of duplicates. There were then a total of 119 matrices. As is common practice with ordering methods, we identify dense rows/columns (similarly to Colamd [10], we identify dense rows as rows/columns having more than $10\sqrt{n}$ nonzeros, for a matrix of size n) at the outset and apply the ordering methods to the remaining rows/columns. The final ordering on the matrix $\mathbf{A}$ is then obtained by putting the dense rows/columns at the end.

We try $\beta \in \{0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$ for the algorithm QCC (Algorithm 1). We present three sets of experiments. In the first set (Section 5.1.1), we identify the best tie-breaking mechanism in QCC. In the second set (Section 5.1.2), we try to find a β that strikes a good balance between partitioning run time and ordering quality. In the third set of experiments (Section 5.1.3), we compare the proposed ordering approaches with the hypergraphs created by QCC(β) and MC37, which are denoted as $\mathcal{H}_{QCC(\beta)}$ and $\mathcal{H}_{MC37}$, with three alternatives: (i) using the clique-covers CC to create hypergraphs [7], denoted as $\mathcal{H}_{CC}$; (ii) AMD [1]; and (iii) MeTiS [29].

5.1.1 Tie-breaking in QCC.

We investigate the tie-breaking mechanisms of QCC in order to choose the best strategy. For each β, we run the algorithm with the tie-breaking mechanism of favouring the nodes with smaller degrees first (SF) for expanding cliques and with the one favouring the nodes with larger degrees first (LF). We then counted how many times each mechanism was better (in the case of ties, the scores of both are incremented) with respect to the number $m_{\mathbf{M}}$ of rows in $\mathbf{M}$ and the number $nnz(\mathbf{M})$ of nonzeros in $\mathbf{M}$. The scores are shown in Table 1. As seen from this table, SF obtained a better score than LF in both metrics for all β in the test set. As β increases, the difference is less marked with often the same values for SF and LF. We see this because the scores add up to a number larger than the number of matrices, 119. For $\beta = 0.5$ there are only three such cases; this number increases with increasing β and is 31 for $\beta = 1.0$. From these experiments, we identify the tie-breaking mechanism of favouring the nodes with smaller degrees first as preferable, especially when β is small, as this yields fewer rows and nonzeros in $\mathbf{M}$.

5.1.2 The parameter β in QCC.

The parameter β affects (i) the run time of the algorithm QCC; (ii) the number of rows of the approximate structural factor $\mathbf{M}$ (this also affects the number of nonzeros of $\mathbf{M}$); (iii) the partitioning time (of the hypergraph partitioning

	$m_{\mathbf{M}}$		$nnz(\mathbf{M})$	
β	LF	SF	LF	SF
0.5	10	112	20	102
0.6	14	108	25	96
0.7	34	113	34	112
0.8	37	112	38	111
0.9	40	110	44	106
1.0	65	85	75	75

Table 1 – The number of matrices (each cell can be at most 119) in which a tie-breaking mechanism, smaller first (SF) or largest first (LF), was the best for differing β.

			time (s.)		
method	$\frac{\text{nrows}(\mathbf{M})}{\text{nrows}(\mathbf{A})}$	$\frac{\text{nnz}(\mathbf{M})}{\text{nnz}(\mathbf{A})}$	StrDcp	PaToH	$\frac{\lvert\mathbf{L}\rvert}{\lvert\mathbf{L}\rvert_{\beta=0.5}}$
$\mathcal{H}_{QCC(0.5)}$	0.44	0.20	0.29	15.75	1.00
$\mathcal{H}_{QCC(0.6)}$	0.81	0.25	0.37	19.51	0.98
$\mathcal{H}_{QCC(0.7)}$	2.10	0.37	0.53	29.82	0.92
$\mathcal{H}_{QCC(0.8)}$	2.29	0.40	0.59	32.94	0.90
$\mathcal{H}_{QCC(0.9)}$	3.09	0.49	0.76	42.31	0.89
$\mathcal{H}_{QCC(1.0)}$	3.58	0.56	0.98	51.19	0.89
$\mathcal{H}_{CC}$	3.36	0.65	0.98	42.77	0.90
$\mathcal{H}_{MC37}$	2.01	0.47	0.69	27.76	0.88

Table 2 – Geometric mean over all matrices. The column StrDcp gives the geometric mean of the run time of different structural decomposition algorithms. The column $\frac{|\mathbf{L}|}{|\mathbf{L}|_{\beta=0.5}}$ is the geometric mean of the fill-in with respect to that of $\beta = 0.5$.

tools applied to $\mathbf{M}$); and (iv) the quality of the final ordering on the matrix $\mathbf{A}$. It is expected that the smaller β is, the faster the QCC algorithm and the smaller the number of rows in the structural factor $\mathbf{M}$, as there will be less quasi-cliques. The quality of the final ordering on the matrix $\mathbf{A}$ is expected to increase when increasing β, since the approximation becomes better. We now observe this expected outcome on the data set.

We present some statistics for all matrices in our data set for differing β in Table 2. In this table, we compare the effect of different structural decomposition methods for creating hypergraphs. As seen in this table, the number of rows of $\mathbf{M}$, i.e., the number of quasi-cliques, increases with increasing β (and also the number of nonzeros of $\mathbf{M}$ increases). Increasing β results in an improved ordering quality with $\mathcal{H}_{QCC(\beta)}$. This however increases the run time for the structural decomposition algorithm QCC and the run time for PaToH, where QCC is always much faster. The β values $\{0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$ resulted in the most fill-in for 78, 23, 8, 0, 4, and 6 cases, respectively. Given these results, we identify $0.7 \leq \beta \leq 0.8$ as a good choice: β values larger than 0.8 offer little improvement in the fill-in with increased run time for the partitioning. For comparison, we also present results with $\mathcal{H}_{CC}$, using the clique cover

	AMD	$\mathcal{H}_{CC}$	$\mathcal{H}_{MC37}$	$\mathcal{H}_{QCC(\beta)}$ with different β					
				0.5	0.6	0.7	0.8	0.9	1.0
num	22	39	50	21	24	33	42	41	41
min	0.11	0.10	0.10	0.11	0.11	0.10	0.10	0.10	0.10
gmean	0.77	0.86	0.87	0.79	0.80	0.83	0.86	0.85	0.85
overall improvement	0.04	0.05	0.06	0.04	0.04	0.05	0.05	0.05	0.05

Table 3 – Performance of different algorithms with respect to MeTiS in terms of fill-in for Cholesky factorization. The rows "num", "min", and "gmean" concern the cases where each method obtains strictly smaller fill-in than MeTiS. The last row shows the improvement in all 119 matrices where the method is used only in the matrices in which it was better than MeTiS, and MeTiS is used in the rest.

(CC) method [7], and also with the proposed $\mathcal{H}_{MC37}$. As seen from the results, among the two exact structural decomposition methods, MC37 is preferable to CC. The sizes of the resulting hypergraphs are smaller for MC37 than when using CC, and the resulting fill-in is less when MC37 is used to obtain an ordering. We defer further comparison of the structural decomposition methods in terms of fill-in to the next subsection.

5.1.3 Fill-in comparison with other methods.

The structural decomposition method should not be applied to all matrices for fill-reducing purposes. Consider, for example, the model problem which corresponds to the 5-point discretization of a 2D domain. In the corresponding graph, the maximum cliques are of size 2, and the best clique cover corresponds to the node-edge incidence matrix. This would create too many cliques. We prefer a small number of cliques or quasi-cliques (with respect to the number of vertices). Furthermore, if quasi-cliques are used, the pattern of $\mathbf{M}^T\mathbf{M}$ should preferably be close to that of $\mathbf{A}$. One way to use structural decomposition methods is to develop some criteria as to when to use them (and to use the state of the art methods, such as MeTiS, in the remaining cases). In preliminary investigations [17], we proposed such recipes for $\mathcal{H}_{CC}$, $\mathcal{H}_{QCC(0.8)}$ and $\mathcal{H}_{MC37}$ where approximately a 2.5% improvement over MeTiS was obtained for each method. The "quality" setting of PaToH tries many algorithms in the multilevel framework to obtain improved results. This makes the quest for finding a recipe difficult. Therefore, our recipe for the use of a structural decomposition method is the ideal recipe: run MeTiS and the proposed method, and then choose the better result. Such ideal recipes are used in actual solvers [8, 15] as poly-algorithms.

Table 3 compares different methods with AMD and MeTiS. In this table, the number of matrices (among 119) in which a method obtained better fill-in than MeTiS is given in the row "num". The minimum ratio and the geometric mean of fill-in with respect to that of MeTiS is given in the next two rows (for the matrices in which better results than those of MeTiS are obtained). By looking at the geometric means (the row gmean in the table), one notices

at least 13% improvement with respect to MeTiS with the proposed methods. However, this should be put into perspective by using all the matrices in the data set (this was our aim in choosing a large set of matrices automatically). To do so, we give the overall improvement over the whole data set in the final row of the table. The overall improvement of a method is computed by using the method when it gives a better result than MeTiS and by using MeTiS on other cases. For example, if $\mathcal{H}_{MC37}$ is used in 50 matrices from the data set (where it obtains better results than MeTiS) and MeTiS is used in the other 69 matrices, we obtain an improvement of 6% with respect to using MeTiS only. As seen in the table, with a good choice of pattern factorization method, improvements of 4% to 6% in the fill-in are possible with respect to MeTiS. Also as seen in the table, the minimum ratio achieved by all methods with respect to MeTiS is around 0.10 (always on the matrix `Sandia/ASIC_680k`). Removing this matrix from the data set yields 2% overall improvement with $\mathcal{H}_{QCC(0.5)}$, 3% overall improvement with AMD, $\mathcal{H}_{CC}$, $\mathcal{H}_{QCC(0.6)}$, $\mathcal{H}_{QCC(0.7)}$, $\mathcal{H}_{QCC(1.0)}$, and 4% overall improvement with $\mathcal{H}_{MC37}$, $\mathcal{H}_{QCC(0.8)}$, and $\mathcal{H}_{QCC(0.9)}$.

5.2 LU factorization

We used matrices satisfying the following properties. They are real, square, of order greater than 10000, have at least 4 nonzeros per row on average, have at most 20 million nonzeros and, as reported in the UFL collection, have a numerical symmetry smaller than 0.85. We exclude matrices recorded as "graph" in the UFL collection, because most of these matrices have nonzeros from a small set of integers (for example $\{-1, 1\}$) and are reducible. Again, these properties are used for automatically selecting a set of matrices from diverse application domains. There were a total of 41 matrices in the UFL collection satisfying these properties. On four of those matrices (with ids 898, 2260, 2265, and 2267) LU factorization took too much time with some of the ordering methods. We removed these matrices and experimented with 37 matrices in total. We first permuted the matrices columnwise with MC64 [13] to have the main diagonal corresponding to a maximum product matching and scaled the matrices such that the diagonal entries were one, and the others no larger than one in magnitude. We then added a diagonal shift (equal to the order of a matrix) to the matrices to ensure strong diagonal dominance. The matrices are then ordered with AMD [1] (run on $\mathbf{A} + \mathbf{A}^T$), Colamd, Hund [25] (on $\mathbf{A}$), and MeTiS (on $\mathbf{A} + \mathbf{A}^T$) and the resulting matrix was factorized with SuperLU [12, 33]. On any given matrix, the fill-in resulting from orderings returned by Colamd was always much worse than the others (this is expected as Colamd reduces a bound on the fill-in that could result from any row interchanges). This was also true for Hund for the same reason (the geometric mean of the ratio of the fill-in due to Hund to that due to MeTiS was 1.88 on the aforementioned data set). We therefore do not include the results with Colamd or with Hund and exclude these two ordering methods from the remaining discussion.

Given the success of $\mathcal{H}_{MC37}$ for the Cholesky factorization, we used MC37 on $\mathbf{A} + \mathbf{A}^T$ to get a structural factor as proposed in Section 3.2.3. This turned out to be better than our bi-quasi-clique cover based heuristics. We therefore present results only with hypergraph partitioning based ordering where MC37 is used to obtain the hypergraphs. The overall algorithm is again denoted by $\mathcal{H}_{MC37}$.

	AMD	$\mathcal{H}_{MC37}$
num	9	23
min	0.61	0.62
geomean	0.85	0.92
overall improvement	0.04	0.05

Table 4 – Performance of different algorithms with respect to MeTiS (with $\mathbf{A}+\mathbf{A}^T$) in terms of fill-in for LU factorization on the matrices where the method obtains strictly less fill-in than MeTiS. The last row shows the improvement over all instances where the method is used only in the cases in which it was better than MeTiS and MeTiS is used in the rest. There are 37 matrices in total.

As was done in the previous subsection, we examine the potential of using the structural decomposition in Table 4. In this table, the number of cases in which a method has obtained better fill-in than MeTiS is given in the row "num". The minimum ratio and the geometric mean of fill-in with respect to that of MeTiS is given in the next two rows. The final row gives the improvement over the whole data set, where a method is used when it gives a better result than MeTiS (that is, for example, $\mathcal{H}_{MC37}$ is used in 23 matrices and MeTiS is used in the other 14 matrices). As seen in the table, with a good choice of a structural decomposition method, an overall improvement of 4% to 5% in the ordering is possible. We note that if $\mathcal{H}_{MC37}$ is used for all 37 matrices, the geometric mean of the ratio to MeTiS is 0.98. In comparison, the geometric mean of the ratio of AMD to MeTiS is 1.16 if we include all 37 matrices.

5.3 QR factorization

We chose a set of 15 rectangular matrices (with ids 261, 799, 981, 1332, 1870, 1871, 1872, 1964, 2025, 2032, 2069, 2112, 2128, 2129, 2134) from the UFL collection. We added to the data set all nine matrices from the LPnetlib collection having more than 9000 rows and columns. On the LP matrices, earlier work by Çatalyürek et al. [7, Table 5.4] indicate that better results are to be expected. We have experimented within the context of Corollary 2. That is, we consider an $\mathbf{M}$ for a given $\mathbf{A}$ such that $\mathbf{A}^T\mathbf{A} \equiv \mathbf{M}^T\mathbf{M}$ holds.

When necessary, we transposed the matrices so that $m \geq n$. We computed $\mathbf{A}^T\mathbf{A}$ and ordered the resulting matrix with MeTiS. In order to get a reasonable ordering and execution time when the matrices have dense rows or columns, we ordered only the non-dense rows/columns of $\mathbf{A}^T\mathbf{A}$ with MeTiS and appended the dense rows to the end of the permutation (as done for AMD and variants). We sparsified the $m \times n$ input matrix $\mathbf{A}$ to obtain $\mathbf{M}$ (so that $\mathbf{A}^T\mathbf{A} \equiv \mathbf{M}^T\mathbf{M}$) and applied PaToH to partition this matrix rowwise into $K = \max(2, \lfloor n/500 \rfloor)$ parts. We then used an SBBD form as the constraint in Ccolamd [8, 10, 11] to find a column ordering for $\mathbf{A}$. This approach is denoted by $\mathcal{H}_s$. We also used MC37 on $\mathbf{A}^T\mathbf{A}$ to obtain $\mathbf{M}$, which is methodologically more comparable to MeTiS in requirements (both require the pattern of $\mathbf{A}^T\mathbf{A}$). This approach is denoted by $\mathcal{H}_{MC37}$. We used the Matlab function `symbfact(A, 'col', 'lower')` to compute the number of nonzeros in $\mathbf{R}$ symbolically. Using `symbfact` and the Colamd results from the UFL collection (available in the data field `amd_rnz`),

	MeTiS	$\mathcal{H}_s$	$\mathcal{H}_{MC37}$
min	0.48	0.55	0.49
max	1.14	1.08	0.98
gmean	0.78	0.89	0.77
min	0.66	0.68	0.67
max	1.13	1.02	1.14
gmean	0.93	0.91	0.90

Table 5 – Performance of different algorithms with respect to Colamd in terms of fill-in for QR factorization on general rectangular matrices (top half of the table) and on the LPnetlib matrices (in the bottom half of the table).

we compare $\mathcal{H}_s$ and $\mathcal{H}_{MC37}$ with MeTiS below. Since on LP matrices we expect a better result, we give results for the two different data sets separately so that we do not skew the results in favour of the proposed methods.

As seen from the top half of Table 5, MeTiS is 22% better than `Colamd` on general rectangular matrices. Using $\mathcal{H}_s$ and $\mathcal{H}_{MC37}$ in combination with `Ccolamd` greatly improves results. MeTiS is only 11% better than $\mathcal{H}_s$ which can be seen as Ccolamd with special constraints, and it is slightly worse than $\mathcal{H}_{MC37}$, which can again be seen as Ccolamd with another set of constraints. The situation improves for the LPnetlib matrices as expected in favour of `Colamd`, $\mathcal{H}_s$ and $\mathcal{H}_{MC37}$. This time, MeTiS is better than `Colamd` by about 7%. Hypergraph partitioning based methods $\mathcal{H}_s$ and $\mathcal{H}_{MC37}$ are better than MeTiS by 2% and 3%, respectively.

The advantage of `Colamd` (with an SBBD form using $\mathcal{H}_s$ or without) with respect to MeTiS and $\mathcal{H}_{MC37}$ is that the product $\mathbf{A}^T\mathbf{A}$ is not needed at all. Even if there is no dense rows in $\mathbf{A}$, the product is denser than $\mathbf{A}$ and forming and storing the product creates a non-negligible overhead.

We now briefly discuss what we gained by using sparsification to realize Corollary 1. Performing sparsification did not cause us to lose any quality (the geometric mean of the fill-in resulting from using the sparsified matrices to that resulting from using the original matrices was 1.01). The geometric mean of the ratio of the execution time of the hypergraph partitioning tool with the sparsified matrices to the original ones was 0.65. When the time spent in the sparsification routine was added before taking the ratios, the geometric mean became 0.68. Therefore, we conclude that with the sparsification method, we gain 32% in run time over the hypergraph partitioning tool without losing the quality of the resulting ordering.

5.4 Evaluation of the results

The tools for obtaining fill-reducing orderings for Cholesky factorization are well developed. In particular, local ordering methods (such as the approximate minimum degree algorithm and its variants), and using graph partitioning based methods to set up constraints in the local ordering methods are well tested and improved over the years. Using hypergraph partitioning methods to obtain desirable forms to define the constraints is a recent approach, promising some advantages over the graph partitioning based method (discussed elsewhere [7,

Section 2.5]). Our methods were demonstrated to be better than the existing hypergraph partitioning based ordering methods. However, they are not consistently better than other existing methods. To us, it is very improbable to obtain consistently better results than the well established tools with a single method. Therefore, we think that it is necessary to try to combine all approaches in a poly-algorithm to obtain the best results. With such an approach, we demonstrated improvements of about 5% and 6% in the Cholesky and LU factorization, indicating that the proposed approaches would be a very useful component in a poly-algorithm.

For QR factorization, the picture is much clearer. Using a hypergraph partitioning method to obtain constraints for `Ccolamd` helps greatly. Using sparsification helps to reduce run time for the hypergraph partitioning tool. The best identified hypergraph partitioning based ordering method obtains better results than MeTiS (while having the same inconvenience of forming and storing $\mathbf{A}^T\mathbf{A}$ while building a structural factor using MC37). This seems to be avoidable by applying MC37's algorithm on an implicitly stored matrix, instead of $\mathbf{A}^T\mathbf{A}$. On a special set of matrices, where a structural factorization already exists, the proposed method using MC37 and the simpler one, which does not form $\mathbf{A}^T\mathbf{A}$, obtain better results than MeTiS. These observations suggest the following: (i) the proposed hypergraph partitioning based ordering methods would again be very useful in a poly-algorithm; (ii) the hypergraph partitioning based ordering that does not need $\mathbf{A}^T\mathbf{A}$ is the method of choice (preferable to Colamd) when computing $\mathbf{A}^T\mathbf{A}$ is prohibitive.

6 Conclusion

We have discussed fill-reducing ordering methods for sparse Cholesky, LU, and QR factorization. Our approach is based on an (approximate) structural decomposition of the given input matrix, where the structural factor is expressed as a hypergraph. The proposed approach generalizes a previous study [7] in finding proper structural factors for Cholesky, and extends the results to the LU factorization. We have argued that for QR factorization, similar methods are applicable, where the overall scheme hinges on the fact that the $\mathbf{R}$ factor is the Cholesky factor of $\mathbf{A}^T\mathbf{A}$.

In all three factorizations, we reported results that are better than MeTiS on some non-negligible number of matrices. Combining with MeTiS, the final averages are improved over 4% for Cholesky and LU factorization with respect to using MeTiS only. In QR factorization, one of the proposed methods based on the graph of $\mathbf{A}^T\mathbf{A}$ obtains results comparable to or better than MeTiS. In QR factorization of a class of matrices, where a structural decomposition already exists, the other proposed method, which never forms $\mathbf{A}^T\mathbf{A}$, obtains better results than MeTiS. In all three factorizations, a structural decomposition or sparsification of $\mathbf{A}$ are demonstrated to lead to a better quality ordering than one based on standard hypergraph partitioning approaches.

The structural decomposition of symmetric matrices with MC37 led to better results than most of the proposed structural decomposition methods. This was especially useful in the case of QR factorization. This encourages us to investigate adapting the algorithm to work on an implicitly stored $\mathbf{A}^T\mathbf{A}$.

Acknowledgements

This work was partially supported by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR), and also by ANR project SOLHAR (ANR-13-MONU-0007). The research of I. S. Duff was supported in part by the EPSRC Grant EP/I013067/1.

References

[1] P. R. Amestoy, T. A. Davis, and I. S. Duff. An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications*, 17(4):886–905, 1996.

[2] G. Ausiello, P. Crescenzi, V. Kann, Marchetti-Sp, G. Gambosi, and A. M. Spaccamela. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer, 1999.

[3] C. Aykanat, A. Pinar, and Ü. V. Çatalyürek. Permuting sparse rectangular matrices into block-diagonal form. *SIAM Journal on Scientific Computing*, 25(6):1860–1879, 2004.

[4] I. Brainman and S. Toledo. Nested-dissection orderings for sparse LU with partial pivoting. *SIAM Journal on Matrix Analysis and Applications*, 23 (4):998–1012, 2002.

[5] Ü. V. Çatalyürek and C. Aykanat. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. *IEEE Transactions on Parallel and Distributed Systems*, 10(7):673–693, Jul 1999.

[6] Ü. V. Çatalyürek and C. Aykanat. *PaToH: A Multilevel Hypergraph Partitioning Tool, Version 3.0*. Bilkent University, Department of Computer Engineering, Ankara, 06533 Turkey. PaToH is available at `http://bmi.osu.edu/~umit/software.htm`,, 1999.

[7] Ü. V. Çatalyürek, C. Aykanat, and E. Kayaaslan. Hypergraph partitioning-based fill-reducing ordering for symmetric matrices. *SIAM Journal on Scientific Computing*, 33(4):1996–2023, 2011.

[8] Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickam. Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate. *ACM Trans. Math. Softw.*, 35(3):22:1–22:14, 2008.

[9] T. A. Davis and Y. Hu. The University of Florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1):1:1–1:25, 2011.

[10] T. A. Davis, J. R. Gilbert, S. I. Larimore, and E. G. Ng. A column approximate minimum degree ordering algorithm. *ACM Transactions on Mathematical Software*, 30(3):353–376, 2004.

[11] T. A. Davis, J. R. Gilbert, S. I. Larimore, and E. G. Ng. Algorithm 836: COLAMD, a column approximate minimum degree ordering algorithm. *ACM Transactions on Mathematical Software*, 30(3):377–380, Sept. 2004.

[12] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. H. Liu. A supernodal approach to sparse partial pivoting. *SIAM Journal on Matrix Analysis and Applications*, 20(3):720–755, 1999.

[13] I. S. Duff and J. Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM Journal on Matrix Analysis and Applications*, 22:973–996, 2001.

[14] I. S. Duff and J. A. Scott. A parallel direct solver for large sparse highly unsymmetric linear systems. *ACM Trans. Math. Softw.*, 30(2):95–117, June 2004. ISSN 0098-3500.

[15] I. S. Duff and J. A. Scott. Towards an automatic ordering for a symmetric sparse direct solvers. Technical Report RAL-TR-200601, Atlas Centre, Rutherford Appleton Laboratory (RAL), Oxon OX11 0QX, 2006.

[16] I. S. Duff and B. Uçar. Combinatorial problems in solving linear systems. In U. Naumann and O. Schenk, editors, *Combinatorial Scientific Computing*, chapter 2, pages 21–68. CRC Press, 2012.

[17] I. S. Duff, O. Kaya, E. Kayaaslan, and B. Uçar. Presentation at Workshop Celebrating 40 Years of Nested Dissection (ND40), available at `http://perso.ens-lyon.fr/bora.ucar/papers/nd40.pdf`, July 2013.

[18] A. Ene, W. Horne, N. Milosavljevic, P. Rao, R. Schreiber, and R. E. Tarjan. Fast exact and heuristic methods for role minimization problems. In *Proceedings of the 13th ACM symposium on Access control models and technologies*, SACMAT '08, pages 1–10, New York, NY, USA, 2008. ACM.

[19] J. M. Ennis, C. M. Fayle, and D. M. Ennis. Assignment-minimum clique coverings. *J. Exp. Algorithmics*, 17:1.5:1.1–1.5:1.17, 2012.

[20] B. O. Fagginger Auer and R. H. Bisseling. A geometric approach to matrix ordering. *CoRR*, abs/1105.4490, 2011.

[21] A. George and M. T. Heath. Solution of sparse linear least squares problems using givens rotations. *Linear Algebra and its Applications*, 34(0):69–83, 1980.

[22] A. George and J. W. H. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Review*, 31(1):1–19, 1989.

[23] A. George, M. Heath, and E. Ng. A comparison of some methods for solving sparse linear least-squares problems. *SIAM Journal on Scientific and Statistical Computing*, 4(2):177–187, 1983.

[24] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, 3rd edition, 1996.

[25] L. Grigori, E. Boman, S. Donfack, and T. Davis. Hypergraph-based unsymmetric nested dissection ordering for sparse LU factorization. *SIAM Journal on Scientific Computing*, 32(6):3426–3446, 2010.

[26] HSL. HSL 2011: A collection of Fortran codes for large scale scientific computation, 2011. `http://www.hsl.rl.ac.uk`.

[27] Y. Hu and J. Scott. Ordering techniques for singly bordered block diagonal forms for unsymmetric parallel sparse direct solvers. *Numerical Linear Algebra with Applications*, 12(9):877–894, 2005.

[28] Y. F. Hu, K. C. F. Maguire, and R. J. Blake. A multilevel unsymmetric matrix ordering algorithm for parallel process simulation. *Computers & Chemical Engineering*, 23(11–12):1631–1647, 2000.

[29] G. Karypis and V. Kumar. *MeTiS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices Version 4.0*. University of Minnesota, Department of Comp. Sci. and Eng., Army HPC Research Center, Minneapolis, 1998.

[30] O. Kaya, E. Kayaaslan, and B. Uçar. On the minimum edge cover and vertex partition by quasi-cliques problems. Technical report RR-8255, INRIA, Feb. 2013.

[31] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. Wiley–Teubner, Chichester, U.K., 1990.

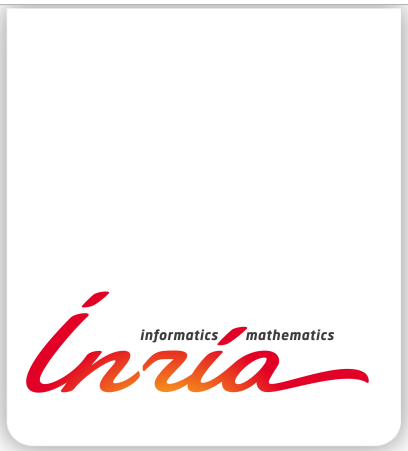
[32] X. S. Li and J. W. Demmel. SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Trans. Math. Softw.*, 29:110–140, June 2003. ISSN 0098-3500.

[33] X. S. Li, J. W. Demmel, J. R. Gilbert, L. Grigori, M. Shao, and I. Yamazaki. SuperLU Users' Guide. Technical Report LBNL-44289, Lawrence Berkeley National Laboratory, September 1999.

[34] J. Orlin. Contentment in graph theory: Covering graphs with cliques. *Indagationes Mathematicae (Proceedings)*, 80(5):406–424, 1977.

[35] B. Uçar, Ü. V. Çatalyürek, and C. Aykanat. PaToH MATLAB interface. `http://bmi.osu.edu/~umit/software.html`„ July 2009.

[36] B. Uçar, Ü. V. Çatalyürek, and C. Aykanat. A matrix partitioning interface to PaToH in MATLAB. *Parallel Computing*, 36(5–6):254–272, 2010.



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399